IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

WISCONSIN ALUMNI RESEARCH FOUNDATION, Plaintiff, v. INTEL CORPORATION, Defendant.	Civil Action No. 3:08-cv-00078-bbc DEMAND FOR JURY TRIAL

DEFENDANT INTEL CORPORATION'S
OPENING MARKMAN BRIEF

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Federal Cases

Defendant Intel Corporation ("Intel") submits this claim construction brief in support of its proposed claim constructions for the patent-in-suit, U.S. Patent No. 5,781,752 ("the '752 patent"). Plaintiff Wisconsin Alumni Research Foundation ("WARF") has asserted claims 1 and 2 of the '752 patent. The '752 patent describes and claims a computer processor that executes instructions in a particular way. A copy of the '752 patent is attached as Tab 1 to accompanying affidavit of Stephen Muller. Attached as Tab 2 is a summary of the parties' respective proposed claim constructions, including constructions for four terms on which the parties agree.

I. BACKGROUND TECHNOLOGY

A. Computers, Processors, and Instructions

1. The basic configuration of a computer

A conventional computer system (e.g., a personal computer) has four main components: (a) a central processing unit, also known as the "processor"; (b) software programs—e.g., applications such as word processors and email—that run on the processor; (c) memory, which stores the programs and their data; and (d) input/output devices such as display screens, keyboards, and printers. (Ex. 4, Clark Decl. ¶ 15.)

The processor is hardware, a microscopic arrangement of electrical circuits. The software programs are a sequence of instructions that the processor executes. The instructions are stored in the memory of the computer, and then "fetched" for execution. The processor hardware carries out—"executes"—the fetched instructions. (Ex. 4, Clark Decl. ¶ 16.)

Each instruction causes the processor to perform a specific task or operation. Each individual task is small (e.g., adding two values to get a sum), but because modern processors execute billions of instructions per second, the computer is capable of accomplishing complex tasks quickly. The functions associated with modern computer performance—e.g., surfing the

Internet; email; word processing; playing music and videos—are rooted in a few basic types of instructions, which are repeated over-and-over at fantastic speeds. (Ex. 4, Clark Decl. ¶ 17.)

2. The three types of computer instructions

Broadly speaking, computer programs have three types of instructions: computation instructions; data instructions; and control instructions. (Ex. 4, Clark Decl. ¶ 18.)

Computation instructions perform arithmetic operations, e.g., addition. (Ex. 4, Clark Decl. ¶ 19.) Examples of computation instructions are found in Instruction 101 and Instruction 102 of the prior art U.S Patent No. 5,555,432 (hereafter "'432 patent"). *See* Tab 3, '432 patent, 1:65-67.[1]

Instruction 101: *A=5+3; and*

Instruction 102: *B=A*9,*

Instruction 101 tells the processor to compute the sum of five plus three. Instruction 102 tells the processor to multiply the sum from Instruction 101 by nine.

Data instructions involve the movement of data. (Ex. 4, Clark Decl. ¶ 19.) For example, after performing the 5+3 computation directed by Instruction 101 of the '432 patent, the processor may be instructed to place the result ("8") in a memory location. This instruction is called a **store** instruction. To perform Instruction 102, the processor would then fetch the value of A (i.e., 8) from the memory location and place it into a specialized memory within the processor called a "register," whereupon the processor can then perform the computation. Fetching data for computation is called a **load** instruction, because the processor "loads" the data

[1] References preceded by "Tab" refer to the exhibits attached to the accompanying affidavit of Stephen Muller.

from memory. The value being loaded from the memory location is one that was previously stored in that memory location by a store instruction. (Ex. 4, Clark Decl. ¶ 19.)

A store instruction changes the data in the memory location, replacing whatever residual value was there with the new value (unless, by coincidence, the old value and newly stored value are the same). Storing is sometimes called "writing" to a memory location. A load instruction does not change the data in the memory location, but merely "reads" the value. (Ex. 4, Clark Decl. ¶ 20.)

Store and load instructions are keyed to particular memory **addresses**, so the processor knows which memory location ("address") to which it must store data, or from which memory location it must load data. (Ex. 4, Clark Decl. ¶ 21.)

As described above, Instructions 101 and 102 from the '432 patent can involve storing and loading data. To perform the above Instructions 101 and 102, the processor:

(i) computes the sum 5+3

(ii) stores the result of the computation (i.e., "8") into a memory location

(iii) loads the value 8 from that memory location into the necessary register

(iv) computes the multiplication 8x9 to achieve the final result ("72").

The third type of instruction is a **control instruction**. A control instruction "jumps" or "branches" the processor from one set of instructions to a different set of instructions. For example, in a word processor program, a control instruction will move the program from one particular subroutine ("display the letters on the screen as she types") to another subroutine ("save the data when she pushes the save button"). (Ex. 4, Clark Decl. ¶ 22.)

Review of key terminology:

term	*basic meaning*
store instruction	a data instruction that stores data to a memory location
load instruction	a data instruction that loads data from a memory location

3. The order in which instructions are executed

Program instructions are written in a particular order, which the '752 patent refers to as "memory order," and are executed in a particular order, which the '752 patent refers to as "program order." "Memory order" means that instructions are written down and stored in a particular, sequential order: a certain instruction is first, and additional instructions follow in logical sequence. "Program order" means that the instructions execute in memory order, except where an instruction causes the program to "jump" or "branch" to a different section of the memory order. In other words, program instructions are written in a particular order and generally execute in that order. (Ex. 4, Clark Decl. ¶ 23; Tab 1, '752 patent, 1:22-49.)

Originally, processors could **only** execute instructions in program order. This meant that the processor executed a given instruction before executing the instruction that appeared immediately after it in program order. This caused logjams: a delay in the execution of an early instruction delayed the execution of the later instructions. (Ex. 4, Clark Decl. ¶ 24; Tab 3, '432 patent, 1:15-26.)

Processors were improved so they could execute instructions **out-of-order**, i.e., when the instruction was "ready for execution," rather than where the instruction appeared in the program order. An instruction is "ready" to execute if the data necessary to perform the instruction (e.g., the values "5" and "3" in Instruction 101), and the necessary hardware resources on the

processor are available. (Ex. 4, Clark Decl. ¶ 25; Tab 3, '432 patent, 1:35-58.) Out-of-order execution thus avoided delays in which one slow instruction could hold up the whole line of later instructions that were "ready" for execution. (Ex. 4, Clark Decl. ¶ 25.)

Review of key terminology:

term	*basic meaning*
program order	instructions generally execute in the order in which they are written
out-of-order execution	allowing an instruction to execute out of program order

4. Dependent and independent instructions

Some instructions must be executed in program order, or will result in an error. (Ex. 4, Clark Decl. ¶ 26.) An instruction that can only be properly executed after another instruction is "dependent" on the earlier instruction. (Ex. 4, Clark Decl. ¶ 26; Tab 3, '432 patent, 2:5-8.) An example is seen in the '432 patent, 1:65-67:

Instruction 101: *A*=5+3; *and*

Instruction 102: *B*=*A**9,

Here, Instruction 101 is **independent**: it performs a computation using a known arithmetic operation ("+") on known data ("5" and "3"). Instruction 101 does not "depend" on any other instruction to feed it information or pre-conditions. (Ex. 4, Clark Decl. ¶ 26.)

However, Instruction 102 is a **dependent** instruction because it must have the result of Instruction 101 before it (Instruction 102) can properly execute. Specifically, until Instruction 101 calculates the value of A, Instruction 102 cannot accurately calculate the value of B. Instruction 102 is thus "dependent" on Instruction 101. (Ex. 4, Clark Decl. ¶ 26; Tab 3, '432 patent, 2:4-11.)

In sum, independent instructions can safely be executed out-of-order. By contrast, dependent instructions cannot safely be executed out-of-order. If an instruction is dependent, the processor must wait to execute it until the instruction on which it depends has been executed. Otherwise, there will be an error. (Ex. 4, Clark Decl. ¶ 27; Tab 1, '752 patent, 2:14-15, "Usually, instructions that are data dependent must be executed in the program order.")

Review of key terminology:

term	***basic meaning***
dependent instruction	an instruction that will cause an error if it is executed prematurely, i.e., before the earlier instruction(s) on which it depends

5. The dependency of load instructions on earlier store instructions

As discussed above, load and store instructions are data instructions. Load instructions are normally dependent on some store instructions of earlier program order. If a load and a store access the **same memory location** for data, the later load instruction will be dependent on the earlier store instruction, for common sense reasons: a value cannot be retrieved from a location if it has not first been put into that location. An example of this is seen in Fig. 2 of the '752 patent and the accompanying text 8:40 through 9:15. (Ex. 4, Clark Decl. ¶ 28.)

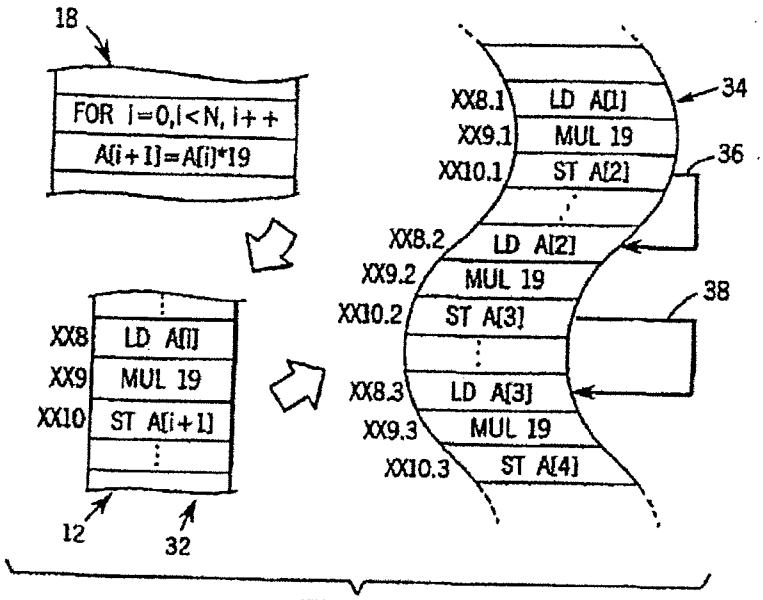

FIG. 2

At element 36 in Fig. 2, the program order expects that store instruction 10.1 will store a value into memory location A[2], and then load instruction 8.2 will later load that value from the same memory location A[2]. The value will then be used in computation instruction 9.2, which multiplies the loaded value by 19. (Ex. 4, Clark Decl. ¶ 29.)

If the store/load program order is reversed (i.e., if load instruction 8.2 executes before store instruction 10.1), there will be an error: the load will retrieve a value from memory location A[2], but the correct value will not yet have been stored there. In fact, the premature execution of load 8.2 would retrieve whatever residual value happened to be residing in A[2], rather than the particular value which store 10.1 was supposed to put there. This wrong value would then be used in computation instruction 9.2 (which multiplies the loaded value by 19), resulting in an incorrect computation. A cascade of errors could follow. (Ex. 4, Clark Decl. ¶ 30.)

In sum, a load instruction is dependent on a previous store instruction if both instructions access the same memory location for data: data must be stored to a location before it can be retrieved from that same location. (Ex. 4, Clark Decl. ¶ 30.) The '752 patent states:

> "Data dependency" is a dependency of instructions that use data [*i.e, load instructions*] on earlier instructions that change the data [*i.e., store instructions*]. These latter [*load*] instructions may correctly execute only if the earlier [*store*] instructions using the same data do not change the common data. . . .

(Tab 1, '752 patent, 1:67 to 2:5.)

Review of key terminology:

term	***basic meaning***
dependent **load** instruction	a load instruction is dependent on a store instruction that (a) operates on the same memory location, and (b) precedes the load in program order

6. Ambiguous dependencies

In the above example of Instructions 101 and 102 from the '432 patent, it is obvious that Instruction 102 is dependent on Instruction 101. In that case, the processor has an easy task: do not execute Instruction 102 out of order, but rather wait for execution of Instruction 101. If all scenarios were so easy, then out-of-order execution would be simple: execute all instructions out of order, except for dependent instructions.

However, a major complication arises because in many instances, the processor cannot readily determine whether an instruction is dependent on another until **after** the processor executes the relevant set of instructions. (Ex. 4, Clark Decl. ¶ 31; Tab 1, '752 patent, 2:18-22.) The '752 patents refers to a dependency as "**ambiguous**" if the processor must execute the instruction to determine whether there is, or is not, a dependency. (Ex. 4, Clark Decl. ¶ 31; Tab 1, '752 patent, 2:5-14.)

The '752 patent provides an example of ambiguous dependency at Table I and 6:20-54. (Ex. 4, Clark Decl. ¶ 32.)

TABLE I

I1	st $M(R_1)$
I2	ld $M(R_2)$
I3	ld $M(R_3)$

Here, instruction "I1" is a store instruction ("st"), and the next instruction in program order, I2, is a load instruction ("ld"). The store instruction I1 stores to the memory location identified by the value that is stored in register R1. Thus, for example, if the value stored in R1 is "10," then the store instruction I1 will store data to memory location 10. The load instruction I2 loads from the memory location identified by the value that is stored in register R2. Thus, for example, if the value stored in R2 is "11," then the load instruction I2 will load data from memory location 11. (Ex. 4, Clark Decl. ¶ 33.)

In some circumstances, R1 and R2 might have the **same** stored value, which means that the I1 store instruction and the I2 load instruction will access the **same** memory address. For example if the values stored in R1 and R2 are both equal to 10, then the load instruction and store instruction will both access memory location 10. In such case, the load I2 is **dependent** on the store I1. If I2 executes before I1, there will be an error, because I2 will try to load data from memory location 10 that I2 has not yet stored there. If this load/store pair is allowed to execute out-of-order, there will be an error. (Ex. 4, Clark Decl. ¶ 34.)

On the other hand, R1 and R2 might have **different** stored values. For example, if the value stored in R1 is 10 and the value stored in R2 is 11, then the store instruction accesses memory location 10 and the load instruction accesses memory location 11. There is no data dependency, and the load instruction may safely execute before the store instruction. In this case, the load and store are independent and do not form a "pair." (Ex. 4, Clark Decl. ¶ 35.)

In these examples, the value of the registers R1 and R2 cannot be discerned from looking at the instruction. Until the instructions are executed, the processor cannot know the values stored in the registers, cannot know the memory locations that are to be accessed, and hence cannot know if there is (or is not) a dependency. (Ex. 4, Clark Decl. ¶ 36.)

Therefore, in this example in Table I, the dependency between the load and the store is **ambiguous**. (Ex. 4, Clark Decl. ¶ 37; Tab 1, '752 patent, Table I and 6:20-54.)

Review of key terminology:

term	*basic meaning*
ambiguous dependency	the possibility that an instruction **might** be dependent on another instruction(s) that appears earlier in program order

7. Speculating

When an instruction's dependency is ambiguous, the processor does not know what will happen if it allows out-of-order execution. If the instruction turns out to be dependent, then out-of-order execution will almost certainly result in an error. Since the processor does not know if the instruction is or is not dependent (i.e., the instruction's dependency is "ambiguous"), the processor faces a choice. (Ex. 4, Clark Decl. ¶ 38.)

The "safe" choice is to wait, and execute the ambiguous instruction in program order. The "risky" choice is to go ahead (i.e., **not** wait) and execute the ambiguous instruction out-of-order, taking the risk that doing so might result in error. Electing to wait slows the overall execution of the program, but avoids errors. Electing not to wait speeds things up, but can cause errors. (Ex. 4, Clark Decl. ¶ 38.)

Executing an ambiguous instruction out-of-order is known as "**speculating**," because the processor does not know whether the result will be correct or incorrect. (Ex. 4, Clark Decl. ¶ 39;

Tab 1, '752 patent, 2:28-30, "some ILP processors may provide for 'speculation,' that is, execution of an instruction that has ambiguous dependency as if it had no dependency at all.")

If the speculation turns out to be wrong—i.e., the instruction causes an error when it executes—this is called a "**mis**-speculation." The '752 patent refers to a mis-speculation as "an instruction that has been executed prematurely and erroneously." (Ex. 4, Clark Decl. ¶ 39; Tab 1, '752 patent, 7:39-41.)

Review of key terminology:

term	*basic meaning*
speculating	allowing an ambiguously dependent instruction to execute before the instruction(s) on which it might depend
mis-speculating	speculative execution that results in an error

8. The prior art

The '752 patent purports to have invented a processor architecture that employs a specific technique to speculatively execute load instructions ahead of store instructions. The '752 patent does not purport to invent—and did not invent—out-of-order execution, speculation, speculation by load/store instructions, or the use of prediction techniques (described below) to make speculation more efficient. These techniques were known in the art before the '752 patent. (Ex. 4, Clark Decl. ¶ 47; *See, e.g.*, Tab 3, '432 patent; Tab 5, '506 patent; Tab 6, Smith article).

a. The prior art allowed speculation for "all" instructions, and then "squashed" instructions that turned out to be wrong

In an early technique from the prior art, a processor that was capable of out-of-order execution would "in general permit loads to be executed ahead of stores." (Ex. 4, Clark Decl. ¶ 40; *See* Tab 5, '506 patent, 1:38-39.) In other words, the processor would allow all ambiguous load instructions to speculate. Allowing all ambiguous loads to speculate caused errors (mis-

speculations), because (as explained above) there are many load instructions that are dependent on store instructions. (Ex. 4, Clark Decl. ¶ 40.)

To account for the errors, the prior art processors employed a "recovery process." The recovery process "typically involves invalidating the load instruction that caused the violation and all newer instructions in program order beyond that load instruction, and [then] reissuing the load instruction." (Ex. 4, Clark Decl. ¶ 41; Tab 5, '506 patent, 1:45-49.) In other words, once the processor learned of the mistake, it invalidated the results and re-executed the instructions, but this time in program order. The '752 patent refers to this recovery process as "squashing," and further refers to executed instructions as being "retired" if they are not squashed. (Ex. 4, Clark Decl. ¶ 41; Tab 1, '752 patent, 2:43-50, out-of-order instructions are "squashed" if the processor learns that speculation did cause an error; 7:31-35, instructions that have executed properly are then "retired.")

Employing a recovery process is less than ideal because it consumes the processor's resources. The prior art '506 patent refers to the recovery process as a "penalty" for mis-speculating. (Ex. 4, Clark Decl. ¶ 42; Tab 5, '506 patent, 1:64-65.) The '752 patent adds, "[s]quashing instructions is a time consuming process that to some extent defeats the advantages to be gained" from speculating. (Tab 1, '752 patent, 2:51-53.)

In sum, one early prior art technique used two steps: (1) allow all ambiguous instructions to speculate, and (2) "recover" the instructions that turn out to have speculated erroneously.

b. The prior art develops "prediction" as a way to avoid the penalty associated with squashing

Unsatisfied with this early "all or nothing" approach to speculation, computer architects began to look for a technique whereby speculation for an ambiguous load instruction was

permitted **some** of the time, but not all of the time. A technique that was developed was called **prediction**.[2] (Ex. 4, Clark Decl. ¶ 43.)

Using prediction, the processor would predict, in advance, the likelihood of a successful speculation, and then allow speculation only upon a prediction that the speculation would probably be successful. (Ex. 4, Clark Decl. ¶ 44; Tab 5, '506 patent, 2:3-7.) If the prediction indicated that the environment was safe, then speculation was allowed: the ambiguous load could execute out-of-order. On the other hand, if the prediction indicated that a violation was "likely to occur," then speculation was dis-allowed. (Ex. 4, Clark Decl. ¶ 44; Tab 5, '506 patent, 1:61 to 2:9.)

The advent of predictors "provided an improved apparatus for permitting load and store instructions [to] issue and execute out of order." (Ex. 4, Clark Decl. ¶ 45; Tab 5, '506 patent 1:61 to 2:1.) With the improvement, some ambiguous load instructions could speculate if they were predicted to be safe, whereas other ambiguous load instructions could **not** speculate because they were predicted to be unsafe. The use of **prediction** as a regulator for **speculation**, thus improved the efficient use of out-of-order execution: speculation could be used in situations where it was probably going to be successful, but avoided in situations where it was probably going to result in an error. Prediction would thus maximize the benefit of speculation (faster execution of programs) while minimizing the costs (the time-consuming recovery process for mis-speculations). (Ex. 4, Clark Decl. ¶ 45.)

---

[2] "Prediction" and "mis-speculation" are terms used in the '752 patent, and the '752 patent claims a "prediction" that has certain specific features with respect to a "mis-speculation." Intel does not suggest that any processor that predicts the likelihood of successful speculation engages in "prediction" regarding a "mis-speculation" as those terms are claimed in the '752 patent.

Thus, prediction systems for load/store instructions pre-dated the '752 patent. For example, the prior art '506 patent describes the following apparatus (*see* Tab 5, '506 patent, Abstract):

> An apparatus to **dynamically controls the out-of-order execution of load/store instructions** by detecting a store violation condition and avoiding the penalty of a pipeline recovery process. The apparatus permits a load and store instruction to issue and execute out of order and incorporates a unique store barrier cache which is used to **dynamically predict whether or not a store violation condition is likely to occur and, if so, to restrict the issue of instructions to the load/store unit** until the store instruction has been executed and it is once again safe to proceed with out-of-order execution.

The discussion above concerns the out-of-order execution of **data** instructions, but these techniques were also used for out-of-order execution of **control** instructions. (Ex. 4, Clark Decl. ¶ 46; *see, e.g.*, Tab 6, Smith, "A Study of Branch Prediction Strategies," ISCA Conference, 1981.)

B. The '752 Patent

The '752 patent—entitled "Table Based Data Speculation Circuit For Parallel Processing Computer"—concerns a specific processor architecture that employs a specific technique to speculatively execute load instructions. (Ex. 4, Clark Decl. ¶ 47.)

In general terms, claim 1 (the asserted, independent claim) claims a processor that detects mis-speculations and prevents data speculation by instructions that have historically mis-speculated at an unacceptably high rate. The processor includes a "data speculation circuit," a "predictor," and "prediction threshold detector," which are described in more detail below. (Ex. 4, Clark Decl. ¶ 48; Tab 1, '752 patent, 14:36-52.)

The (alleged) central insight of the '752 patent is that, "The present inventors have recognized that most data dependence mis-speculations can be attributed to a few static STORE/LOAD instruction pairs . . . . These particular instruction pairs can be identified based

on previous mis-speculations." (Ex. 4, Clark Decl. ¶ 49; Tab 1, '752 patent, 3:51-62. *Accord id.* 14:15-20, because a "relatively limited number of LOAD/STORE pairs will create mis-speculation," the invention concerns developing a "list of critical LOAD/STORE pairs.")

In other words, the named inventors concluded that only a few load/store pairs were causing most of the mis-speculations. Thus, according to the inventors, load instructions could speculate freely while these few "bad apple" pairs were identified and isolated. (Ex. 4, Clark Decl. ¶ 50.)

The '752 patent describes the following basic sequence (Ex. 4, Clark Decl. ¶ 51):

1. Ambiguous load instructions are allowed to execute out-of-order, i.e., to speculate.

> (Tab 1, '752 patent, 3:64-66, "If there is no history of data mis-speculation, an instruction is executed without further inquiry."; *see also id.* 11:22-23.)

2. A "data speculation circuit" tracks each mis-speculation committed by a load instruction.

> (Tab 1, '752 patent, 7:4-7, "The data speculation circuit is responsible for . . . detect[ing] any mis-speculations." *See also id.* 3:67 to 4:4; 4:12-17.)

3. Load instructions that mis-speculate are "squashed" and re-executed "at a later time." (Tab 1, '752 patent, 7:39-48.)

4. A "predictor" looks at the "history of mis-speculations" and tracks which load/store pairs speculated correctly versus incorrectly. (Tab 1, '752 patent, 3:67 to 4:7)

5. A "threshold detector" shuts off speculation for load/store pairs whose historical rate of mis-speculations reaches a certain threshold. (Tab 1, '752 patent, 4:21-23; 14:1-6)

The '752 patent describes a particular mechanism that performs this basic sequence. Specifically, in the event that a load instruction mis-speculates, the "data speculation circuit"

sends a "mis-speculation indication" to the "predictor" circuit. (Ex. 4, Clark Decl. ¶ 52; Tab 1, '752 patent, 9:64-67, if it detects a mis-speculation, the data speculation circuit sends a "signal" to the predictor so that the predictor can "adjust[] its prediction.")

Having received the mis-speculation indication, the predictor enters the offending load instruction and its corresponding store instruction (i.e, the store instruction on which the load instruction depends) into a table as a pair. (Ex. 4, Clark Decl. ¶ 52; Tab 1, '752 patent, 11:42-45, prediction table indicates whether a data dependence exists "between a LOAD/STORE pair"; 12:64-66, table is checked "to see whether the LOAD/STORE pair causing the mis-speculation is in the prediction table.")

The table entry itself includes three pieces of information: the identity of the load instruction, the identity of the store instruction, and the prediction value (described below) for that pair of instructions. (Ex. 4, Clark Decl. ¶ 52; Tab 1, '752 patent, 11:9-13.)

The prediction table is shown as element 44 at Fig. 5. Element 109 is the prediction value. In Fig. 5, "1" is the prediction value for the load/store pair LD8/ST10. (Ex. 4, Clark Decl. ¶ 53.)

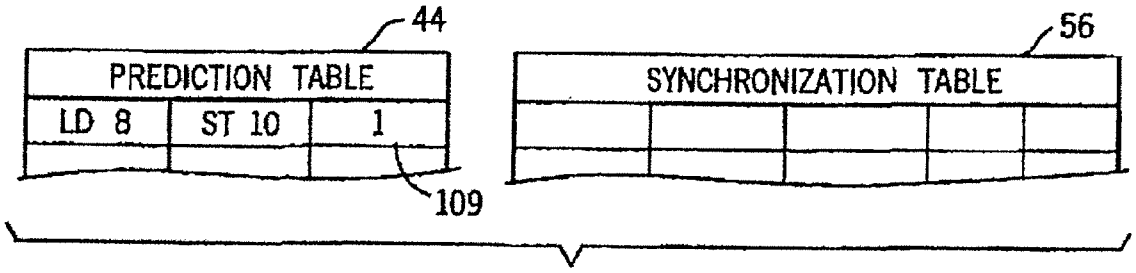

FIG. 5

The table typically starts with a prediction value of zero. Each time that speculative execution of that load/store pair results in a mis-speculation, the count goes up. The count can go down in other circumstances, for example if the store instruction in the pair in fact executes

before the load instruction in the pair. (Ex. 4, Clark Decl. ¶ 54; Tab 1, '752 patent, 11:33-35, the prediction value starts at zero and is then "incremented and decremented"; Fig. 4 element 120 (decrementing value if the store has been executed first); Fig. 7 element 214 (incrementing value), and 12:14-17; 8:8-11, the prediction "is updated based on historical mis-speculations detected by the data speculation circuit.")

The higher the prediction (i.e., the higher the value in the table), the greater the likelihood that the load instruction in the load/store pair will mis-speculate the next time it is executed:

> [T]he prediction is used to determine the likelihood of a dependency between two instructions in the future. **The higher the prediction the more likelihood of mis-speculation** if the instruction in the first column is executed before the instruction in the second column.

(Ex. 4, Clark Decl. ¶ 55; Tab 1, '752 patent, 14:1-6.)

In other words, the prediction is a mis-speculation counter: a mis-speculation makes the count go higher. The higher the mis-speculation count, the more likely it is that a future speculative execution of that load/store pair will result in error.

If the mis-speculation count (i.e., the prediction) in the table entry for that load/store pair ultimately rises to a certain value, the risk of mis-speculation by that load/store pair is deemed to be too high. Then, a circuit called the "prediction threshold detector" prevents that load/store pair from speculating in the future. The patent refers to the triggering count as a "threshold" and/or as a value "within a predetermined range." (Ex. 4, Clark Decl. ¶ 56; Tab 1, '752 patent, 4:21-23; 14:1-6.)

In other words, the processor of the '752 patent: (a) allows load/store pairs to speculate, (b) maintains a mis-speculation counter for a load/store pair that has mis-speculated, (c) the counter goes up when the speculation by that load/store pair is unsuccessful, and (d) disables

speculation for that load/store pair if the counter reaches a certain upper threshold. (Ex. 4, Clark Decl. ¶ 57.)

Accordingly, the following "plain meaning" may be ascribed to Claim 1 of the '752 patent (Ex. 4, Clark Decl. ¶ 58):

'752 patent, claim 1	*"Plain English" summary*
1. In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having **a data speculation circuit for** detecting data dependence between instructions and **detecting a mis-speculation** where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is in fact executed before the data producing instruction, a data speculation decision circuit comprising:	A "**data speculation circuit**" detects mis-speculations by a load/store pair, i.e., detects when a dependent load instruction is "in fact executed" before the store instruction on which it depends. (The parties agree that a "data consuming instruction" is a load instruction and a "data producing instruction" is a store instruction.)
(a) a **predictor** receiving a mis-speculation indication from the data speculation circuit to produce a **prediction associated with the particular data consuming instruction** and based on the mis-speculation indication; and	Upon detecting the mis-speculation, the data speculation circuit sends an "indication" to the **predictor**. In response to the 'indication," the predictor produces a "prediction," i.e., a mis-speculation count for that particular load/store pair.
b) a prediction threshold detector **preventing** data speculation for instructions having a prediction **within a predetermined range**.	If the count reaches a certain "threshold" (i.e., moves "within a predetermined range" of values), then a **prediction threshold detector** prevents speculation for that load/store pair

C. Additional concepts

There are a number of other features discussed in the '752 patent specification, but Intel does not presently believe them to be relevant to a disputed Markman issue. Four that bear mention are the synchronization table, "instances" of load/store pairs, additional reasons to increment or decrement the prediction value, and alternative means of managing the prediction value.

The '752 patent describes a "synchronization table" that is used in conjunction with the prediction table to disable speculation by load/store pairs. (E.g., Fig. 4, elements 106 through 122 and accompanying text, 11:42 to 12:19.) This table does not appear in the asserted claims, and concerns implementation-level details that Intel presently believes are not relevant to the parties' disputes.

The '752 patent notes that the same load/store pairs are sometimes executed repeatedly—over and over again—in a loop. The patent states that the repetition creates different "instances" of the same instruction set, and discusses how such instances may be dealt with in the context of mis-speculation. (8:62 to 9:7.) The word "instances" does not appear in the asserted claims.

The patent describes additional reasons for incrementing or decrementing the prediction value. (E.g., Fig. 9, elements 308 and 316 and accompanying text, 13:5-17, 20-30; and Fig. 7, element 214 and accompany text, 12:50-55.) Intel believes that these details are not relevant to the parties' dispute.

The '752 patent also notes that there are "other" methods for calculating a prediction value. (14:6-14.) Again, Intel believes that these details are not relevant to the parties' disputes.

II. LAW

A. Basic rules of claim construction

The claims of a patent define the invention, and it is an "evasion of the law" to construe a patent in a manner different from the plain import of its terms. *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312 (Fed. Cir. 2005) (*en banc*) (internal quotation omitted).

Claim terms are generally given the "ordinary and customary meaning . . . that the term would have to a person of ordinary skill in the art in question at the time of the invention." *Phillips*, 415 F.3d at 1313.

The Federal Circuit has set forth a hierarchy to be employed to review both the intrinsic and other, "extrinsic," evidence as part of the claim construction process. "The best source for understanding a technical term is the specification from which it arose, informed, as needed, by the prosecution history." *Phillips*, 415 F.3d at 1315 (internal quotation and edit omitted). The specification is "the single best guide," and "[u]sually, it is dispositive." *Id.* In particular, "in case of doubt or ambiguity," the specification is relied upon for "ascertaining the true intent and meaning of the language employed in the claims." *Id.* (internal quotation omitted). The specification may reveal a "special definition" of a claim term that differs from an otherwise plain meaning. *Phillips*, 415 F.3d at 1316. If so, then, "the inventor's lexicography governs," regardless of what the ordinary meaning might otherwise be. *Id.* A claim term may be defined by lexicography in the specification, "without an explicit statement" to redefine it. *Honeywell Int'l, Inc. v. Universal Avionics Sys. Corp.*, 493 F.3d 1358, 1362 (Fed. Cir. 2007).

The prosecution history (which is part of the "intrinsic" evidence that includes the patent itself) is also considered during claim construction. *Phillips*, 415 F.3d at 1317. Although the file history "often lacks the clarity of the specification," it nevertheless, "can often inform the meaning of the claim language by demonstrating how the inventor understood the invention." *Phillips*, 415 F.3d at 1317.

Extrinsic evidence—such as technical dictionaries—is "less significant" but nevertheless "useful" if used properly. *Phillips*, 415 F.3d at 1317, 1322 ("Dictionaries or comparable sources are often useful to assist in understanding the commonly understood meaning of words and have been used both by our court and the Supreme Court in claim interpretation.").

B. Restricting the scope of claims

It is improper to limit a patent claim to the examples or embodiments described in the specification, or to read a limitation from the specification into the claims. *See Phillips*, 415 F.3d at 1323.

However, there are instances in which the embodiments "define the outer limits of the claim term" rather than being merely "exemplary in nature." *Phillips*, 415 F.3d at 1323. This rule is similar to the basic proposition of claim construction: "The patent system is based on the proposition that claims cover only the invented subject matter." *Phillips*, 415 F.3d at 1321.

At least two applications of this rule are relevant to this case. First, the scope of patent claims generally cannot exceed a patentee's statement in the specification as to what constitutes "the invention." *See, e.g., Inpro II Licensing SARL v. T-Mobile USA, Inc.*, 450 F.3d 1350, 1355 (Fed. Cir. 2006) (claims cannot "enlarge what is patented beyond what the inventor has described as the invention"); *Alloc, Inc. v. ITC*, 342 F.3d 1361, 1368-69 (Fed. Cir. 2003) (limiting claims to what the specification described as "the invention").

Second, it is appropriate to limit an invention to a disclosed embodiment if the specification, taken as a whole, makes clear that the invention should be so limited. Such limitation can be proper even if nothing in the specification or prosecution history acts as an explicit disclaimer of subject matter. *See, e.g., Nystrom v. Trex Co., Inc.*, 424 F.3d 1136, 1143-46 (Fed. Cir. 2005) (relying upon the specification to give a narrow construction to the term "board" despite the absence of any explicit disclaimer of subject matter).

III. DISPUTED TERMS

As discussed in depth below, the parties' disputes center on three aspects of the (alleged) invention:

1. The processor identifies and prevents mis-speculation based on **pairs** of load/store instructions, i.e., load and store instructions that form a pair because they access the same memory location.

2. The processor identifies and prevents mis-speculation by looking to actual instances of mis-speculation, i.e., historical instances in which instructions have "**in fact executed**" and led to error.

3. The processor begins by allowing instructions to speculate, and keeps permitting speculation until the instructions prove to be unreliable by making **multiple** mistakes.

These concepts are described in the patent specification as being part of the invention, and are reflected in the claim language itself. Intel's proposed constructions follow the teachings of the specification and the plain import of the claims.

WARF's proposals seek to alter and expand the claim by focusing on constructions that are based on theoretical features, and not related to the problems and solutions identified and discussed in the '752 patent.

The purpose of claim construction is to identify "the invented subject matter" of the claims (*Phillips*, 415 F.3d at 1321), to "capture the scope of the actual invention" (*id.* at 1324), and to interpret the claims so as to cover "what the inventors actually invented" (*id.* at 1316, internal quotation omitted). Intel's proposed constructions follow that rule, WARF's proposed constructions do not.

Support for the proposition that Intel's proposed claim constructions are, in fact, how the terms would be interpreted by a person of ordinary skill in the art are found at Tab 4, Clark Decl. ¶¶ 61-86.

A. "in fact executed"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
1. In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having a data speculation circuit for detecting data dependence between instructions and detecting a mis-speculation where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is **in fact executed** before the data producing instruction, a data speculation decision circuit comprising:	**in fact executed:** a LOAD instruction is "in fact executed" before the STORE instruction when the LOAD instruction has actually accessed or was certain to access data that has not yet been updated by the STORE instruction.	**in fact executed**: a load instruction is "in fact executed" when the load instruction actually has loaded data from a memory location

Intel submits that the term "in fact executed" should be addressed first, because this term appears in the constructions of a number of the other disputed terms. Intel's proposed construction for "in fact executed" is correct because it follows the plain dictionary definition of the term as confirmed by the patent specification. By contrast, WARF's proposal is contrary to both the plain meaning and the patent specification.

Claim 1 recites a "data consuming" instruction (a) that is dependent on a "data producing" instruction of earlier in program order, and (b) that is "in fact executed" before that "data producing" instruction. The parties agree that a load instruction is a "data consuming" instruction and that a store instruction is a "data producing" instruction. Thus, the claim (as simplified to use the agreed-upon meaning of "data consuming" and "data producing") recites a load instruction that is "in fact executed" before a store instruction on which it depends.

"Execute" is a word with a plain meaning found in technical, computer dictionaries:

> • "execute. To carry out an instruction, process, or computer program." (Tab 7, IEEE Standard Computer Dictionary (1990).)
>
> • "execute: To perform the execution of an instruction or of a computer program. execution: The process of carrying out an instruction or the instructions of a computer program by a computer." (Tab 8, Prentice Hall's Illustrated Dictionary of Computing (2d Ed. 1995).)
>
> • "execute: To run a program, carry out a command, or perform a function." (Tab 9, Dictionary of Computer Words (1995).)

As reflected in these definitions, the program executes an instruction when it "performs" or "runs" through a "process." In fact, a computer program instruction contains a number of steps, and its execution is the process of carrying out these individual steps.

Claim 1 claims the **past tense** of the verb, namely "executed." Thus, it claims an instruction that is finished with the process of execution, i.e., whose execution is completed. An instruction is executed when all its component steps have been "carried out." For emphasis, claim 1 adds extra language "in fact" in front of "executed" to remove any doubt. Thus, the plain meaning of "in fact executed" means "actually carried out."

The specification of the '752 patent expressly discloses when a **load** instruction has been "actually carried out." A load instruction is actually carried out ("in fact executed") when the instruction loads the value from the designated memory location. The '752 specification describes the following sequence for how a sequence of load/store instructions is executed:

> The LOAD instruction 8.1 then **loads the contents of memory location A[1]**. The STORE instruction 10.1 then stores a value in memory location A[2]. The LOAD instruction 8.2 then **loads a value from** the same memory location A[2]. . . . If instruction 8.2 were to be **executed** prior to instruction 10.1, it would [be] operating on erroneous data.

(9:8-15.[3]) Thus, a load instruction executes when it "loads the contents" of a specified "memory location." *Accord* 7:18-20 (indicating that an instruction "which requires **data to be obtained from memory**" is "completely executed" when it has obtained that data); 2:36-40 ("Data speculation, for example, might involve **reading from memory** to obtain data for a later instruction, even though there are earlier stores to that memory location that have not yet been completed. . . ."); 3:40-42 ("any instruction **loading data from memory** can be data dependent on any previous instructions that writes to memory").

The specification also refers to an instruction as "executed" when it has "executed as far as possible" (7:24). At this point, the instruction is "executed" because the only thing that remains is for the instruction to be either squashed (if the processor determines that the execution caused or followed a mis-speculation) or retired (if the processor determines execution did not cause a mis-speculation). *See* 7:31-65 (after execution is completed, the instruction is either squashed or retired).

Accordingly, Intel's proposed construction for "in fact executed" is consistent with the plain meaning of the term "executed" and confirmed by the specification itself.

WARF's proposal is essentially the same as Intel's, except that it adds a new and incorrect concept to the end, namely that a load instruction is executed when it "was certain to access data." This extra language does not appear in the patent specification or in the claim.

Initially, it is not clear what WARF means by the words, "was certain to access data." During the meet-and-confer process leading to the Markman briefing, WARF indicated its view that there are circumstances during the execution of a load instruction when the processor can

---

[3] Unless otherwise noted, all patent citations in Section III of this brief are to Tab 1, the '752 patent.

know the memory location from which it will load, but such loading has not yet occurred. In WARF's view, this is the point in the execution process when the load instruction is "in fact executed."

If this is WARF's argument, the argument is incorrect for two reasons. First, WARF's extra language contradicts the unambiguous language of the claim itself: "in fact executed." By its own terms, this limitation excludes instructions that are still in the process of executing. WARF's construction would re-draft the claim language "in fact executed" into something akin to "almost executed" or "about to be executed."[4]

Second, nothing in the patent specification identifies this "almost executed" scenario. The specification is clear that a load instruction is "executed" when it loads the value from the memory location. (9:8-15; *see also* 7:18-20, 2:36-40.) The specification says nothing about a load being "executed" when it is "certain to access" a memory location. Based on the parties' pre-briefing meet-and-confer discussions, WARF apparently intends to rely on extrinsic evidence to support this extra limitation. But extrinsic evidence cannot expand the meaning of a claim term beyond the plain meaning contemplated by the patent specification. *See, e.g., Nystrom*, 424 F.3d at 1145 (a claim term may not be given a broader meaning "simply because it may be found in a[n] . . . extrinsic source").

---

[4] Other portions of the specification use additional verbiage to refer to the process of executing an instruction. For example, Fig. 3, element 74 recites "issue load request." *See also* 10:49-51 ("the program branches to the previously described process block 74 where the load request is **issued**"); 10:24 ("the program proceeds immediately to process block 74 and a load request is **generated**"); and, 13:53-54 ("this particular load instruction which has now been **released** for execution"). When the specification uses the terms like "load request is issued," "load request is generated" and "released for execution" this means that the execution is in process. Thus, for example, a load instruction that is "released for execution" (13:54) is now in the process of executing.

Finally, WARF's proposed construction is incorrect because it omits the fact that a load instruction operates on a "memory location." This concept is implicit in WARF's proposed language "access data," but such concept should be made explicit: a load accesses data that is in a memory location. It is error to omit this last part.

Thus, Intel's proposed construction is correct because it follows the plain meaning of "execute" and is faithful to the patent specification. WARF's proposal is incorrect because it introduces a new concept—"was certain to access data"—that is contrary to the claim language itself, to the plain meaning of "execute," and to the patent specification.

B. "data speculation circuit"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
1. In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having a **data speculation circuit** for detecting data dependence between instructions and detecting a mis-speculation where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is in fact executed before the data producing instruction, a data speculation decision circuit comprising:	**data speculation circuit:** a circuit that detects data dependence between LOAD and STORE instructions and tracks execution of such instructions in order to detect any mis-speculations arising from the data speculative execution of LOAD instructions.	**data speculation circuit:** a circuit that detects data dependence between load/store pairs and detects a mis-speculation by a load instruction that is in fact executed

The parties agree on the construction of "data speculation circuit" to the extent that it is a circuit that detects data dependence and mis-speculations involving load and store instructions.

There are two main differences in the parties' proposed constructions:

1. Intel seeks a construction that would explicitly state that the detected data dependence is between a "pair" of load/store instructions. This feature is present in the claim language, and is described in the specification as the central insight that led to the '752 patent.

2. WARF omits the claim language "in fact executed" from its proposed construction while at the same time adding unnecessary verbiage.

1. the "data speculation circuit" is limited to a circuit that detects load/store "pairs"

Intel proposes to define "data speculation circuit" as a circuit that detects data dependence between a "load/store pair." WARF proposes to define "data speculation circuit" as a circuit that detects data dependence between "load and store instructions."

Including "pairs" in the Markman construction is necessary for three reasons: (a) the plain meaning of the claim language is that the two instructions under consideration are a load/store pair, (b) the specification states that "the invention" is identifying—and preventing the speculative execution of—load/store pairs that cause repeated mis-speculations, and (c) there is no evidence in the specification that the inventors contemplated a subject matter different than the load/store pair scenario.

a. The plain meaning of the claims

The claim language demonstrates that the instructions under consideration are a pair of instructions, specifically a load instruction that is data dependent on a store instruction.

The preamble of claim 1 identifies a load instruction ("data consuming instruction") and a store instruction ("data producing instruction"). (14:40-41.) The claim language further says that the load instruction is "dependent for its data on a [store] instruction of earlier program order," and that the load has executed "before **the** [store] instruction." (14:41-43.) This language means that the load instruction and the store instruction are a "pair" as that term is used throughout the specification.

These references in the preamble are the antecedent basis for the terms that follow in the body of the claim: the "mis-speculation indication" is an indication that this particular pair has caused a mis-speculation. Indeed, the specification refers to a load/store pair as the thing that

causes mis-speculation: "one load/store pair causes a data mis-speculation" (3:55); "the prediction table is checked to see whether the load/store pair causing the mis-speculation is in the prediction table already" (12:65-67). Finally, in the last limitation of claim 1, the "instructions having a prediction" are the very same pair that was introduced in the preamble.

Thus, the preamble introduces a load instruction that is dependent on a store instruction, what the specification uniformly calls a "load/store pair." The body of the claim follows this pair through the various circuits. Although the word "pair" does not appear in the claim itself, the plain reading of the claim is that it is directed to a load/store pair that has caused an error via mis-speculation.

b. The specification says that "pairs" are the key to the invention

"Pairs" should be included in the definition of "data speculation circuit" because the specification states that this feature is part of "the invention." The "Brief Summary of the Invention" section of the specification (3:50 to 5:34) is directed solely at load/store pairs. At least three times, the specification states that the animating idea of the (alleged) invention is to identify the load/store pairs that mis-speculate, and then prevent them from speculating:

> • **The present inventors have recognized** that most data dependence mis-speculation can be attributed to a few static load/store **pairs** [and] if one load/store **pair causes a data mis-speculation** at a given point in time, it is highly likely that a later instance of the **same pair** will soon cause another mis-speculation. . . . These particular instruction **pairs** can be identified based on previous mis-speculations. (3:51-62; this quotation comes from the section entitled "Brief Summary of the Invention.")
>
> • Specifically, **the present invention provides** . . . a prediction associated with the particular data producing/consuming instruction **pair**. (4:8-20; this quotation comes from the section entitled "Brief Summary of the Invention.")

> • **The present inventors believe** that a relatively limited number of load/store **pairs** will create mis-speculation and so the operation described above prevents the majority of the load/store **pairs** from being slowed in execution. The list of critical load store **pairs** is prepared dynamically . . . . (14:15-21.)

These repeated references to load/store pairs are not surprising, because data speculation involves more than just a load instruction. Data speculation occurs when a particular instruction that uses data (a load) is executed out of order before another particular instruction that may set the value for that data (a store). (1:67 to 2:4.)

Accordingly, the claim must be construed to reflect "the invention" as defined by the inventors in the specification—that is, a circuit that detects data dependence between a "load/store pair." *See, e.g., Inpro*, 450 F.3d at 1355-56 (limiting "host interface" to a "direct parallel bus interface" where specification repeatedly described this as a "very important feature" of "the invention"); *Honeywell Int'l, Inc. v. ITT Indus., Inc.*, 452 F.3d 1312, 1318-19 (Fed. Cir. 2006) (limiting "fuel injection system component" to a "fuel filter," where, "[o]n at least four occasions" the specification referred to a fuel filter as the "invention"; further noting, "The public is entitled to take the patentee at his word and the word was that the invention is a fuel filter."); *Microsoft Corp. v. Multi-Tech Sys., Inc.*, 357 F.3d 1340, 1348-49 (Fed. Cir. 2004) (restricting the terms "sending," "transmitting," and "receiving" to communications that take place on a telephone line—to the exclusion of the Internet—"[i]n light of those clear statements in the specification that **the invention** . . . is directed to communications 'over a standard telephone line'") (emphasis added).

c. The specification shows that the inventors did not contemplate a broader invention

The specification also demonstrates that the load/store pair mechanism is the limit of the invention, not merely one embodiment. There is nothing in the intrinsic evidence that indicates

that the inventors contemplated—let alone intended to claim—a processor that controlled data speculation by anything other than with load/store pairs.

As noted above, the specification states that the named inventors "have recognized" that data mis-speculations were attributable to "a few static store/load instruction **pairs**" that "cause[] a data mis-speculation." (3:51-55.) The named inventors stated that "the present invention" (4:8) concerned mis-speculation by a "data producing/consuming instruction **pair**" (4:20). Accordingly, the specification states that the discovery, and the resulting (alleged) invention, were tied to the load/store pair.

Furthermore, the load/store pair mechanism is the only embodiment disclosed in the specification. Every figure and every description is directed towards identifying and disabling a mis-speculating load instruction **that is paired with a particular store instruction**. Examples from the specification include:

- Fig. 5 and accompanying text 11:8-14 (the entry in the prediction table is for a load instruction, a store instruction, and their joint prediction value); 9:8-15 (the pair that is entered into the table access the same memory location A[2])

- Fig. 4, element 106 ("sync. table entry exists for the particular load/store pair")

- Fig. 7, element 204 (same)

- Fig. 9, element 301 ("is there a pred[iction] table entry for this load/store pair?") and accompanying text 12:65-67 ("[T]he prediction table is checked to see whether the load/store pair causing the mis-speculation is in the prediction table already.")

- 14:3-6 (mis-speculation occurs if "the instruction in the first column is executed before the instruction in the second column")

- 11:30-33 (same)

- 14:19-21 ("The list of critical load/store pairs is prepared dynamically in a synchronization method")

Therefore, the load/store pair is not merely one embodiment of the '752 patent, but rather is a necessary feature of the (alleged) invention. *See, e.g., Watts v. XL Systems, Inc.*, 232 F.3d 877, 882-83 (Fed. Cir. 2000) (with respect to claims concerning connecting sections of pipe, limiting "sealingly connected" to "structure utilizing misaligned taper angles," where this feature was described as part of "[t]he present invention" and was the only disclosed embodiment); *Curtiss-Wright Flow Control Corp. v. Velan, Inc.*, 438 F.3d 1374, 1379 (Fed. Cir. 2006) (limiting claim term concerning an "adjustable" seat to a seat that was adjustable "during operation" of the larger machine, where, "the specification of the [patent-in-suit] consistently, and without exception, describes adjustment that occurs during operation"); *Honeywell v. Universal Avionics*, 493 F.3d at 1362 (in a claim concerning landing an airplane, construing "heading"—against its plain meaning—to mean "bearing," where the specification "discloses no other form of alignment" and also described such alignment as an "important feature of the present invention").

2. The distinction between "load/store pair" and "load and store instructions" is important

At first blush, it may seem that Intel's proposed construction (detecting data dependence between a "load/store pair") and WARF's proposed construction (detecting data dependence between "load and store instructions") are the same. They are not.

The purpose of the '752 patent is to detect data dependence between a **particular** load and a **particular** store (a "pair"), so that data mis-speculation by that particular pair can be prevented. Put another way, the purpose of the invention is not to detect mis-speculation by a load instruction per se, but rather to detect—and thereby ultimately prevent—mis-speculation by a load instruction when it appears with a particular store instruction. It is not the load instruction (or the store instruction) that is being identified for its own sake; rather, it is the two together. Similarly, data speculation is not being prevented for the load instruction generally, but for the load instruction when it appears with the store instruction.

Consider this set of instructions:

Instructions executed	*mis-speculation?*
load A before store X	Yes
load A before store Y	No
load B before store X	No
load B before store Y	Yes

In this example, there are only two scenarios ("pairs") that cause an error: executing "A" ahead of "X"; or, "B" ahead of "Y." The pairs A-Y and B-X do not cause an error. According to the named inventors, there is thus no point in identifying "A" as a problem instruction by itself, nor "X" by itself.

It is when these instructions **are together** ("paired") that there is a problem. The whole purpose of the invention is to disable speculative execution of the **pair** A-X, not A or X individually. And the reason is obvious: when A is paired with Y, there is no problem. When X is paired with B, there is no problem. So, the invention is not to detect dependences between "instructions" generally, but to detect dependence between instruction **pairs** in particular:

> The present inventors have recognized that most data dependence mis-speculations can be attributed to a few static STORE/LOAD instruction pairs. Furthermore, these instruction pairs exhibit "temporal locality" that is, if one LOAD/STORE pair causes a data mis-speculation at a given point in time, it is highly likely that a later instance of the same pair will soon cause another mis-speculation.

(3:50-57.)

Indeed, the specification discourages the attempt to generally identify mis-speculating loads: "[A]ny instruction loading data from memory can be data dependent on any previous instructions that writes [*sic*] to memory. Consequently, predicting and tracking data dependencies . . . **can easily become overwhelming**." (3:39-43.)

Intel's proposed construction thus is consistent with the stated purpose of the invention, whereas WARF's proposed construction eliminates it. *See Phillips*, 415 F.3d at 1316

("Ultimately, the interpretation to be given a term can only be determined and confirmed with a full understanding of what the inventors actually invented and intended to envelop with the claim.") (internal quotation omitted).

3. The second half of WARF's proposal adds unnecessary words

The parties agree that the second half of the construction of "data speculation circuit" concerns detecting mis-speculation by a load instruction. Intel's proposed construction is more direct and more true to the actual claim language: "detects a mis-speculation by a load instruction that is in fact executed." Of course, in Intel's proposed construction, the load instruction is explicitly part of a "pair," as discussed above. The specification speaks alternatively between a load instruction that mis-speculates and a load/store pair that mis-speculates. Both phraseologies mean the same thing: a load instruction in a pair executes before the store instruction in the pair, thus causing a mis-speculation.

WARF's proposed construction is incorrect because it omits "in fact executed" from its construction, thereby deleting an important limitation from the claim. The claim says that the data speculation circuit detects a mis-speculation where a load instruction "is in fact executed." It would be error to delete this language.

Intel does not perceive anything else affirmatively inaccurate in the remainder of WARF's proposed construction—"tracks execution of such instructions in order to detect any mis-speculations arising from the data speculative execution of LOAD instructions." For example, the specification does describe the data speculation circuit as "keeping track" (7:5) of the order of execution so that it can "detect any mis-speculations" (7:7). But these extra words lengthen the construction without conferring benefit, and create the possibility of jury confusion.

In sum, WARF's extra language is at best surplusage, and should be omitted in favor of Intel's more direct and simple proposed construction.

C. "mis-speculation"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
1. In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having a data speculation circuit for detecting data dependence between instructions and detecting a **mis-speculation** where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is in fact executed before the data producing instruction, a data speculation decision circuit comprising:	**mis-speculation:** where a LOAD instruction, that is dependent for its data on a STORE instruction appearing earlier in program order, is in fact executed before the STORE instruction.	**mis-speculation:** where an instruction has been in fact executed prematurely and erroneously

As described in the Background section above at page 10, an instruction mis-speculates when it executes out-of-order, and thereby causes an error. In other words, a mis-speculation is a speculation that turns out to be wrong.

Intel's proposed construction for "mis-speculation" is correct because it reiterates the plain meaning of the term. Indeed, Intel's proposed construction adheres precisely to the specification, which states: "mis-speculation occurs in an instruction that has been executed prematurely and erroneously." ('752 patent, 7:39-41.) Intel's proposed construction is word-for-word from the specification, with the only addition being two extra words—"in fact"—that appear in the claim itself. *See also* 2:66 to 3:8 ("mis-speculation" occurs in an instruction that has "produced an erroneous result").

Since the specification defines the term "mis-speculation," that definition controls. *See Phillips*, 415 F.3d at 1316 ("the inventor's lexicography governs").

Moreover, the lexicography from the specification communicates the concept of mis-speculation in a way that will be meaningful to the jury. An instruction mis-speculates because two things occur. First, a speculation occurs. That is, the instruction executes in advance of an instruction on which it may depend ("prematurely"). Second, mis-speculation then occurs when the prematurely executed instruction turns out to have been dependent on the earlier instruction. The result is that the instruction causes an error ("erroneously"). That is what mis-speculation is, and the concept is plainly disclosed in the specification. Intel's proposed construction is thus correct.

By contrast, WARF's proposed construction fails to clearly communicate to the jury either of these defining features: the premature and erroneous execution of an instruction. WARF's proposal also is incorrect because it is too complicated, drawing-in other elements of the claim language that are not part of the definition of "mis-speculation." References to "program order" and to load/store instructions should be omitted.

D. "where a data consuming instruction . . . is in fact executed before the data producing instruction"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
1. In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having a data speculation circuit for detecting data dependence between instructions and detecting a mis-speculation **where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is in fact executed before the data producing instruction**, a data speculation decision circuit comprising:	**where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is in fact executed before the data producing instruction:** where a LOAD instruction, that is dependent for its data on a STORE instruction appearing earlier in program order, has actually accessed or was certain to access data that has not yet been updated by the STORE instruction	**where a data consuming instruction dependent for its data on a data producing instruction of earlier program order, is in fact executed before the data producing instruction**: where a load instruction that depends on a store instruction has actually loaded data from a memory location before the store instruction has put data into that same memory location

There are no new disputes between the parties with respect to this claim term. Intel submits that its construction is more direct and easier to understand—for example, Intel substitutes "depends on a store instruction" for the longer "dependent for its data on a STORE instruction occurring earlier in program order." But there are no new substantive issues. Instead, the substantive issues presented are the same as above, namely a disagreement over the meaning of "in fact executed."

E. "predictor"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
(a) a **predictor** receiving a mis-speculation indication from the data speculation circuit to produce a prediction associated with the particular data consuming instruction and based on the mis-speculation indication; and	**predictor:** a circuit that receives a mis-speculation indication from the data speculation circuit to produce a prediction	**predictor:** a circuit that receives a mis-speculation indication from the data speculation circuit to produce a prediction **based on historical mis-speculation indications**

The parties agree on the first part of the definition of "predictor." It is a circuit that receives a mis-speculation indication from the data speculation circuit and produces a prediction. (The meaning of "prediction" is discussed in the next section, below.) The parties differ because Intel seeks to include how "predictor" is defined in the specification, namely that the predictor evaluates "historical mis-speculation indications."

Intel's proposed construction is correct because the specification defines a predictor that bases its prediction on historical mis-speculations, and because during prosecution, the inventors emphasized this feature when distinguishing prior art.

In the "Brief Summary of the Invention," the patent specification describes the use of a "predictor based on **the past history of mis-speculations**" to "determine whether the instruction should be executed or delayed." (4:1-4.) Furthermore, during prosecution of the patent, the Examiner rejected claim 1 over the '432 patent (also known as the "Hinton" patent). (*See* Tab 10, Office Action Summary, pp. 3-4.) In response, the named inventors admitted that Hinton "addresses the problem of data dependence by the use of a predictor," but distinguished **their** predictor from Hinton's on the following grounds: "In contrast in the present invention, the

predictor does not determine when data will be available but whether there is data dependence by examining **previous instances of mis-speculation**." (*See* Tab 11, First Amendment, p. 2.)[5]

Thus, in both their summary of "the invention" and in distinguishing prior art in the prosecution history, the inventors stated that the predictor of their invention based its prediction on historical mis-speculations. It is thus necessary to limit the claim term "predictor" to meet that definition. *See, e.g., Watts*, 232 F.3d at 882-83 (limiting "sealingly connected" to "structure utilizing misaligned taper angles," where this feature was described as part of "[t]he present invention" and where the presence of this feature was used during prosecution to distinguish prior art); *Alloc,* 342 F.3d at 1368-70 (in a patent concerning floor panels, construing "locked element" and "locking member" to include the extra limitation "play," where this extra feature was described as part of the invention and where the specification distinguished prior art that did not have this feature).

Additionally, while the issue will be discussed further below in the context of "prediction," it bears mention now that the prediction created by the predictor is based on identifying "data dependencies on an on-going or dynamic basis." (4:31-37.) This means that the prediction produced by the predictor is based on a history of multiple mis-speculations. Thus, the meaning of "prediction" (discussed below) reinforces the proper definition of "predictor" as a circuit that looks to "historical mis-speculation indications" to create its prediction.

Finally, it appears that WARF actually agrees with the foregoing analysis, in that WARF does include the disputed phrase, "based on historical mis-speculation indications" within its

---

[5] The named inventors distinguished Hinton on a number of other grounds not discussed here.

definition of the longer term (discussed below), "prediction associated with the particular data consuming instruction and based on the mis-speculation indication." Given that WARF agrees with the substance of the issue, it is unclear why WARF will not agree that "based on historical mis-speculations" should be included in the definition of "predictor." As noted above, in both the specification and file history, the named inventors used "historical mis-speculations" to define "predictor." *See* 4:1-4 (the invention includes "a predictor based on the past history of mis-speculations"); Tab 11, First Amendment, p. 2 (the predictor in the "present invention" examines "previous instances of mis-speculation").

F. "mis-speculation indication"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
(a) a predictor receiving a **mis-speculation indication** from the data speculation circuit to produce a prediction associated with the particular data consuming instruction and based on the mis-speculation indication; and	**mis-speculation indication:** an indication that the data speculative execution for a LOAD instruction was incorrect.	**mis-speculation indication:** an indication that an instruction has been executed prematurely and erroneously

The data speculation circuit detects a mis-speculation, and sends a resulting "indication" to the predictor circuit. Claim 1(a) recites that the predictor "receive[s] a **mis-speculation indication** from the data speculation circuit."

The parties essentially agree that a mis-speculation "indication" is an indication that there has been a mis-speculation. The parties' disagreement over "mis-speculation indication" thus follows from their disagreement (discussed above) over the meaning of "mis-speculation."

G. prediction"

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
(a) a predictor receiving a mis-speculation indication from the data speculation circuit to produce a **prediction** associated with the particular data consuming instruction and based on the mis-speculation indication; and	**prediction:** a dynamic multi-bit value which indicates the likelihood that the data speculative execution of a LOAD instruction will result in a mis-speculation	**prediction:** a value indicating the likelihood that data speculative execution of a load/store pair will result in a mis-speculation

When it receives the mis-speculation indication from the data speculation circuit, the predictor produces a "prediction." "Prediction" is just a word for a value entered into the prediction table. The value reflects the history of mis-speculations and, hence, indicates the likelihood of a future mis-speculation. As explained above in the Background section of this brief at pages 13-17, the prediction value moves up based on unsuccessful speculations, and the higher the prediction value, the greater the likelihood of a mis-speculation. On this, the parties agree.

The parties differ in two respects. First, Intel's construction provides that the prediction is not for a particular load instruction per se, but rather for a particular load instruction that is part of a "load/store pair." This is the same issue discussed above. One point bears re-emphasis, however. The specification states that, "As will be understood from this description, **the prediction** is used to determine the likelihood of a dependency between **two instructions** in the future." (14:1-6.) This passage is tantamount to lexicography: it defines the prediction. Notably, the definition refers to "two instructions." As the specification makes clear, these are not two random instructions, but rather are two instructions that have been identified as instructions that **together** cause a mis-speculation (e.g., 12:66-67, "the load/store pair causing the mis-speculation"; 3:55, the "load/store pair causes a data mis-speculation"). Accordingly, to

construe the term consistently with the named inventors' lexicography, the term "prediction" must include reference to the load/store pair.

Second, WARF seeks to add an extra limitation, namely that the prediction be a "dynamic multi-bit" value. There is no basis on which to import this extra limitation into the claim.

"Multi-bit" appears to be a limitation invented by WARF for the purposes of litigation. The term does not appear in the specification. Furthermore, "multi-bit" indicates the number of bits that would be needed to express the value of the prediction. The number of bits is an implementation-level detail; the claims of the '752 patent are not written at that level of detail. There is no basis on which to create this new limitation, and then import it into the claim.

The term "dynamic" does appear throughout the specification. For example, "it is one object of the invention to provide a predictor circuit that may identify data dependencies on an on-going or **dynamic** basis." (4:31-33.) The problem is that the specification never defines what "dynamic" means, so importing that word into the claim does little, by itself, to clarify the claim's meaning.

As indicated by the previous quotation ("on-going or dynamic"), the patented processor is designed **not** to disable speculation on the first instance of a mis-speculation. Initially, speculation is allowed for all instructions. (E.g., 3:64-66, "if there is no history of data mis-speculation, an instruction is executed without further inquiry.") The processor allows speculation to continue until it has identified multiple mis-speculations on an "on-going or dynamic" basis. In the patent, the value of the prediction starts at zero. (11:33.) As operation continues, the value of the prediction then moves up and down. (E.g., Fig. 4 element 120 (decrementing the value if the store instruction is executed first); Fig. 7, element 214

(incrementing the value).) When the value of the prediction is low, the likelihood of error due to mis-speculation is low; when the value of the prediction is high, the likelihood of error is high. (E.g., 11:27-33; 14:1-6.) It is only when a predetermined threshold is crossed (e.g., the prediction value passes "5"), that the risk of mis-speculation is determined to be unacceptably high, and speculation is disabled. (4:42-47.) In this way, the prediction is "on-going or dynamic" because the prediction counter moves up and down, and because speculation is allowed until the value hits the threshold. *Accord* 7:67 to 8:3 ("The prediction circuit provides a dynamic indication to the data speculation circuit as to whether data speculation should be performed."); 11:33-35 (the prediction value starts at zero and is then "incremented and decremented").

If that is what WARF means by "dynamic," then the parties agree. But adding the word "dynamic" to modify "prediction" does not necessarily convey this full meaning.

In sum, the term "dynamic" is itself vague. Adding it, by itself, to the claim construction of "prediction" does not convey the full significance of its meaning. And, of course, WARF may have a different meaning in mind that it has not disclosed.

H. "prediction associated with the particular data consuming instruction . . ."

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
(a) a predictor receiving a mis-speculation indication from the data speculation circuit to produce a **prediction associated with the particular data consuming instruction and based on the mis-speculation indication**; and	**prediction associated with the particular data consuming instruction and based on the mis-speculation indication:** a dynamic multi-bit value associated with the particular LOAD instruction which indicates the likelihood that the data speculative execution of that LOAD instruction will result in a mis-speculation, and which is based on historical mis-speculation indications	**prediction associated with the particular data consuming instruction and based on the mis-speculation indication**: a value associated with the particular load instruction that indicates the likelihood that data speculative execution of a load/store pair will result in a mis-speculation, and which is based on historical mis-speculation indications

There are no new disputes between the parties with respect to this claim term. The issues presented are the same as above, namely a disagreement over the limitations "dynamic multi-bit" and "pair."

I. "a prediction threshold detector . . ."

'752 Claim Language	WARF's Proposed Construction	Intel's Proposed Construction
b) a prediction threshold detector preventing data speculation for instructions having a prediction **within a predetermined range**.	**whole phrase:** a circuit that prevents data speculation for a particular LOAD instruction when the prediction associated with said LOAD instruction is in a predetermined range. **range:** a set of values that a quantity or function may assume	**whole phrase**: a circuit that prevents data speculation of a load/store pair if the prediction value for the pair is within a predetermined range. **range**: [AGREED]

The final element of claim 1 concerns the operation of a "prediction threshold detector." As explained above, instructions are executed out-of-order; a data speculation circuit detects mis-speculations and sends a corresponding "indication" to a predictor; the predictor turns the "indication" into a "prediction"; the prediction indicates the likelihood that the instruction (pair) will mis-speculate in the future.

There are no significant new disputes with respect to this portion of the claim. The parties agree as to the construction of "data speculation," "speculation," and "range." *See* Tab 2 (summary of parties' proposed constructions).

The parties continue to dispute the "pair" issue, and that is reflected in their proposed constructions for this term. The only other dispute concerns WARF's proposal to change the claim language "**within** a predetermined range" to "**in** a predetermined range." Intel does not perceive a substantive difference here, but there is no reason to change the language in the claim.

IV. CONCLUSION

For the foregoing reasons, Intel respectfully submits that the Court adopt Intel's proposed claim constructions for the '752 patent.

Richard Bolton
BOARDMAN LAWFIRM LLP
State Bar No. 1012552
One South Pinckney Street, Fourth Floor
Madison, WI 53703
Tel: (608) 283-1789
Fax: (608) 283-1709
rbolton@boardmanlawfirm.com

Of Counsel:

William F. Lee (admitted *pro hac vice*)
Donald R. Steinberg (admitted *pro hac vice)*
Stephen M. Muller (admitted *pro hac vice*)
Victor F. Souto (admitted *pro hac vice*)
WILMER CUTLER PICKERING
HALE AND DORR LLP
60 State Street
Boston, Massachusetts 02109
Tel: (617) 526-6000
Fax:(617) 526-5000

Attorneys for Defendant and Counterclaimant
INTEL CORPORATION

Dated: June 26, 2008

CERTIFICATE OF SERVICE

I hereby certify that on June 26, 2008, copies of the foregoing document were served on the following attorneys at the addresses and in the manner indicated:

BY EMAIL AND FEDERAL EXPRESS:

Michelle Marie Umberger
Lissa Koop
Heller Ehrman LLP
One East Main Street
Ste. 201
Madison, WI 53703-5118

Robert T. Haslam
Anupam Sharma
Heller Ehrman LLP
275 Middlefield Road
Menlo Park, CA 94025

/s/ Richard L. Bolton

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

WISCONSIN ALUMNI RESEARCH FOUNDATION, Plaintiff, v. INTEL CORPORATION, Defendant.	Civil Action No. 08-C-78-C

CERTIFICATE OF SERVICE

I hereby certify that on June 27, 2008, I electronically served upon the Court for filing and uploading via the CM/ECF system of the United States District Court, Western District of Wisconsin, Defendant Intel Corporation's Opening Markman Brief, with amended signature page, in the above matter to go to Attorney Michelle Umberger at michelle.umberger@hellerehrman.com via a notice of electronic filing.

Dated this 27th day of June, 2008.

BOARDMAN, SUHR, CURRY & FIELD, LLP
By:
/s/ Richard L. Bolton
Richard L. Bolton, State Bar No. 1012552
Attorneys for Plaintiffs
1 S. Pinckney Street, 4th Floor
P. O. Box 927
Madison, WI 53701-0927
PH: (608) 257-9521 FX: (608) 283-1709
rbolton@boardmanlawfirm.com